React

What is React

1. React is JavaScript Library.
2. The main focus is building UI as fast as possible.
3. So this is used to Single page application.
4. Means complete website in single page.
5. Created by Jordan Walke at Facebook.

React ek JavaScript library hai jo user interfaces banane ke liye use hoti hai, specifically single-page applications ke liye. Iska development Meta (Facebook) ne kiya tha, aur ab ye ek popular open-source library hai.

When React.js first time used in Facebook?

- News feed

Why React is Fast ?

- React use Virtual Dom

Ques 1. What is React, and why is it used?

React is a JavaScript library for building user interfaces. It is used for creating reusable components and efficiently updating the user interface using the virtual DOM.

React ke Features:

- Component-Based Architecture:

React applications chhoti-chhoti components ka collection hoti hain. Har component apna logic aur UI handle karta hai.

- Virtual DOM:

React directly DOM pe kaam karne ke bajaye Virtual DOM ka use karta hai, jo app ko fast aur efficient banata hai.

- Declarative UI:

React declarative approach use karta hai, jo ki code ko simple aur readable banata hai.

- JSX (JavaScript XML):

React me JSX use hota hai jo HTML aur JavaScript ko ek saath likhne ka option deta hai.

- Reusable Components:

Ek baar component banake multiple jagah use kar sakte ho, jo development fast karta hai.

Package.json

- It is very important file in React which holds all the information about your project(like name, version, dependencies, libraries,etc.)
- It contains metadata about the project, dependencies, scripts and configurations.
- scripts -> Commands for development(dev), production build(build) and preview.
- dependencies -> packages required for the app to run(React, ReactDOM, etc)
- devDependencies -> development-only packages

Difference between package.json & package-lock.json

package.json

- This file is primarily used for managing and documenting metadata about the project, including its name, version, author, dependencies, scripts and other configuration details.

package-lock.json

- This file is generated and updated automatically by npm when installing or updating pakages. It is used to lock the exact versions of dependencies installed in the project.

Component in React js

- A piece of code that can reuse
- Such as function
- But it more powerful than function
- Header, Footer is best example

Component Types

React components can be primarily categorized into two main types:

1. Functional Components (JS function that accept props as argument and return React element)
2. Class Components (ES6 classes that extend React.Component)

Ques 2. What is the difference between functional components and class components?

- Functional components are simple JavaScript functions that return JSX.
- Class components are ES6 classes that extend React.Component and can use lifecycle methods.

Rules for creating Functional Component

1. Should return JSX

```
function HelloWorldComponent() {
return <h3>Hello World</h3>;
    }
```

2. Component name must start with uppercase letter.

Class Component

Example:-

```
import React,{Component} from 'react'
class User extends Component(or React.Component)
{
     render(){
         return (
        <h1>Hello from  User </h1>
        )
     }
}

export default User
```

JSX

- JSX stands for JavaScript XML(JavaScript Syntax Extension).
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.

Ques 3. What is JSX? Can we use React without JSX?

JSX is a syntax extension for JavaScript that allows writing HTML-like code in JavaScript. Yes, React can be used without JSX, but JSX makes the code more readable and concise.

```
const element = React.createElement(
  'h1',      // The type of element (tag name)
  null,      // Props (attributes of the element)c
  'Hello, React without JSX!' // Children (content inside the element)
);
```

Ques 4. How browser understand JSX

Browsers can't read JSX directly. Babel converts JSX into normal JavaScript using React.createElement(). React then creates a virtual copy of the web page, and ReactDOM updates the real page efficiently. This makes JSX work smoothly in the browser.

Ques 5. What is babel in React?

Babel ek JavaScript compiler hai jo aapko modern JavaScript (ES6+ aur baad ki versions) ko older JavaScript versions me convert karne me madad karta hai. Iska main purpose ye ensure karna hai ki aapka JavaScript code har tarah ke browser (including purane browsers) me chal sake.

Kaise Kaam Karta Hai Babel?

Input Code:

Aap ka modern JavaScript/JSX code.

```
const greet = () => {
  console.log("Hello, World!");
};
```

Transpile:

Babel is code ko older JavaScript syntax me convert karta hai.

```
var greet = function () {
  console.log("Hello, World!");
};
```

React JSX ke liye Babel ka Example:

Input:

```
const App = () => <h1>Hello, Babel!</h1>;
```

Transpile:

```
const App = function () {
  return React.createElement("h1", null, "Hello, Babel!");
};
```

State in Functional Component

State ek JavaScript object hota hai jo component ke andar data store karta hai aur uske changes ko track karta hai. Jab state update hoti hai, toh component re-render hota hai.

USE:-

Hum state ka use isliye karte hai kyunki hum react me variable ke through data update nhi kar sakte aur state data ko update karne ke liye ush component ko re-render karta hai

A state of a component is a variable that hold some information that may change over the lifetime of the component.
Jab state badalti hai, React component automatically re-render hota hai.
Functional components me useState Hook ka use hota hai

```
import { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);  // count = state variable, setCount = update
function

  function increment() {
    setCount(count + 1);  // State update
  }

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </>
  );
}

export default Counter;
```

State in Class Component

Class components me this.state aur this.setState() ka use hota hai.

```
import React, { Component } from "react";

class Counter extends Component {
  constructor() {
    super();
    this.state = { count: 0 };  // State initialization
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });  // State update
  };

  render() {
    return (
      <>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </>
    );
  }
}

export default Counter;
```

Props with functional component

Props (short for "properties") are a way to pass data from a parent to a child component. They are read-only.
Props can be objects, arrays, functions, or components
Props can have default values

Example:-

```
function Greeting(props){
  return <h1>Hello, {props.name}! </h1>
}
function App(){
  return <Greeting name="John" />
}
```

Props with class component

Props are passed to a class component same way as functional components.
In class components, props are accessed using this.props

Ques 6. How props are differ from state?

Props are used to pass data from parent to child components and are immutable (cannot be changed inside the component). State is local to a component, mutable, and can be updated using setState(class) or useState(functional).
Props are controlled by the parent, while the state is managed within the component itself.

Hide, Show and Toggle in React

```
import {useState} from 'react'

function App(){
   const [status, setStatus] = useState(true)

   return (
        <div>
         {
         status?<h1>Hello World!</h1>:null
         }

         <button onClick={()=>setStatus(false)}>Hide</button>
         <button onClick={()=>setStatus(true)}>Show</button>
         <button onClick={()=>setStatus(!status)}>Toggle</button>

        </div>
)
}
```

Ques 7. What is the virtual DOM, and how does it differ from the real DOM?

The virtual DOM is a lightweight copy of the real DOM. React updates the virtual DOM first and then syncs it with the real DOM, improving performance

Ques 8. render() methods takes two argument

ReactDOM.render(element, container);

➢ How it Works Internally:

a. Virtual DOM Creation:

When you pass a React element to ReactDOM.render(), React first converts it into a virtual DOM object (a lightweight representation of the real DOM).

b. Real DOM Synchronization:

React compares the virtual DOM with the current state of the real DOM (if it's already rendered). It then calculates the minimal set of changes required to update the real DOM and applies those changes efficiently.

c. Event Binding:

React also sets up event listeners (e.g., for onClick) on the DOM elements during this process.

Conditional Rendering

In react generally we can't use if else or if elseif instead of this we use ternary operator like:
if else - condition ? "ture" : "false"
if elseif else - condition ? "true" : elseif condition ? "true" : "false"

Life Cycle in React

In React, a component goes through different stages from when it is created to when it is removed from the page. React provides special functions that let developers run code at specific moments, like when the component starts, updates, or disappears.
(React component lifecycle refers to the different stages a component goes through from its creation to its removal).

Why is the React Lifecycle Important?

The React Lifecycle is important because it helps manage how a component behaves throughout its existence. Here's why it matters in detail:

1. Manages Component Behavior
   - ❖ Every React component goes through different stages:
     - ➢ Mounting (Creation) – When the component appears on the screen.
     - ➢ Updating – When the component's data (state/props) changes.
     - ➢ Unmounting (Destruction) – When the component is removed.
   - ❖ React provides lifecycle methods (like componentDidMount, componentDidUpdate, componentWillUnmount) to run specific code at each stage.
   - ❖ This is useful for setting up or cleaning up things when a component is created or removed.

2. Optimizes Performance
   - ❖ Lifecycle methods control unneccesary re-renders and help optimize performance.
     - ➢ Example: shouldComponentUpdate() prevents re-renders if the data hasn't changed.
   - ❖ React only updates what's needed instead of refreshing the entire page, making apps faster.

3. Handles Side Effects
   - ❖ Side effects are actions like API calls, event listeners, or timers that happen outside the normal UI updates.
   - ❖ Without proper handling, these can cause memory leaks or unwanted behavior.
   - ❖ Lifecycle methods like componentDidMount and useEffect() (in functional components) ensure:
     - ➢ API calls happen only when needed.

- Event listeners are added and removed correctly.
- Cleanup happens when a component is removed to prevent memory leaks.

4. Enhances Debugging

- ❖ Lifecycle methods help developers track component updates and find issues.

  Example:
  - componentDidUpdate() can log when and why a component re-renders.
  - componentWillUnmount() can be used to clean up resources before a component disappears.
- ❖ This makes debugging easier and prevents issues like infinite loops, memory leaks, or slow performance.

React Lifecycle Phases

There are three main phases in a React component's lifecycle:

1. Mounting Phase (Component Creation)

When a component is being created and inserted into the DOM.

Method	Purpose
constructor()	Initializes state and binds methods
static getDerivedStateFromProps(props, state)	Syncs state with props before rendering.
render()	Returns JSX to be displayed in the UI (Three times update, first when render create or load and second and third time when state or props update)
componentDidMount()	Runs after the component is inserted into the DOM (useful for API calls). Only one time load.

- Execution Order:

constructor() → getDerivedStateFromProps() → render() → componentDidMount()

2. Updating Phase (Component Re-rendering)

Occurs when a component re-renders due to state changes or new props.

Method	Purpose
static getDerivedStateFromProps(props, state)	Syncs state with props before rendering.
shouldComponentUpdate(nextProps, nextState)	Determines if re-rendering is needed (optimization).
render()	Returns updated UI based on new state/props.
getSnapshotBeforeUpdate(prevProps, prevState)	Captures information before DOM updates (useful for animations or scroll position).
componentDidUpdate(prevProps, prevState, snapshot)	Runs after the component updates (used for API calls, logging, etc.).

➢ Execution Order (When State Updates):

getDerivedStateFromProps() → shouldComponentUpdate() → render() → getSnapshotBeforeUpdate() → componentDidUpdate()

3. Unmounting Phase (Component Removal)

When a component is removed from the DOM

Method	Purpose
componentWillUnmount()	Cleanup before the component is removed (remove event listeners, clear timers, etc.).

➢ Execution Order:

componentWillUnmount() (before component is removed)

React Hooks Equivalent (For Functional Components)

With functional components, lifecycle methods are replaced by the useEffect hook.

Class Component	Functional Component (Hooks)
componentDidMount()	useEffect(() => { ... }, [])
componentDidUpdate()	useEffect(() => { ... }, [dependencies])
componentWillUnmount()	useEffect(() => { return () => { ... }; }, [])

➢ Execution Order:

- ❖ useEffect(() => {}, []) → Runs once (equivalent to componentDidMount()).
- ❖ useEffect(() => {}) → If you don't provide a dependency array, `useEffect` runs **after every render**.
- ❖ useEffect(() => {}, [count]) → Runs on state updates (equivalent to componentDidUpdate()).

- ❖ Cleanup function return () => {} → Runs when the component unmounts (equivalent to componentWillUnmount()).

Hooks in React

With hooks, we can use class component features in functional component such as state, life cycle, pure component(a React component that optimizes performance by avoiding unnecessary re-renders).
Hooks React ke special functions hote hain jo functional components me state aur lifecycle methods ka use karne ki facility dete hain. Functional components originally stateless the, lekin hooks ke aane ke baad, hum functional components me bhi state aur side effects ko handle kar sakte hain.

Hooks Kyu Use Hote Hain?

1. Hooks ke bina, state aur lifecycle methods ka use sirf class components me hota tha. Hooks ke aane ke baad, functional components bhi fully capable ho gaye.
2. Functional components hooks ke saath zyada simple aur readable hote hain, compared to class components.
3. Hooks ki madad se hum custom hooks bana kar logic ko reuse kar sakte hain.
4. Hooks ke aane ke baad bhi, existing class components pe koi asar nahi padta.

Commonly Used Hooks

1. **useState**: Component ke andar state manage karne ke liye.
2. **useEffect**: Side effects (jaise API calls, DOM manipulation) handle karne ke liye.
3. **useContext**: Context API ke saath data share karne ke liye.
4. **useReducer**: Complex state logic ke liye, jaise Redux me hota hai.
5. **useRef**: DOM elements ko directly access karne ke liye.

1. useState:-

useState React ka ek hook hai jo functional components ke andar state management ke liye use hota hai. Iska kaam ek stateful value aur usko update karne ke liye ek function provide karna hai.

Syntax :

```
const [state, setState] = useState(initialValue);
```

where, **state**: Ye current state ki value ko represent karta hai.
setState: Ye ek function hai jo state ko update karta hai.
initialValue: State ka initial value jo aap set karte hain.

a. Jab component render hota hai, useState initial value ko use karke ek stateful variable create karta hai.
b. Jab state change hoti hai (via setState), component re-render hota hai aur naye state value ke saath update hota hai.

Note:-

React ke functional components me state changes directly UI ko update nahi karte. Iske liye aapko state management ka use karna hota hai, aur React me ye useState hook ke through kiya jata hai.
(React ka rendering mechanism sirf tab trigger hota hai jab state ya props change hote hain. Aapka counter ek normal variable hai, state nahi, isliye React re-render nahi karta)

2. useEffect:-

React me useEffect kya hota hai?
React ka useEffect ek hook hai jo functional components me side effects handle karne ke liye use hota hai (ya extra kaam karne ka option deta hai).
Iska kaam React ke bahar hone wale operations (side effects) ko control karna hai.
Yeh ek tarike ka lifecycle method hai jo specific situations me code ko execute karne ki permission deta hai, jaise:

- Component render hone ke baad.
- State ya props ke update hone par.
- Component unmount hone par cleanup ke liye.

Side effects ka matlab hai aise operations jo component ke render cycle ke bahar hote hain, jaise:

- API call karna (jaise data fetch karna).
- Event listener add karna (jaise resize ya scroll).
- Timer/interval chalana.
- Document title update karna.

Syntax:-

```
useEffect(() => {
      // Yeh kaam tab chalega jab component render ya update hoga.
return () => {
     // Cleanup ka kaam, jab component hata diya jaye (unmount ho).
 };
}, [dependencies]);
```

useEffect kaise kaam karta hai?

useEffect ke paas 2 parts hote hain:

a. Effect: Yeh part wahi kaam karega jo aap karwana chahte hain (jaise data fetch karna ya kuch update karna).
b. Cleanup (optional): Agar koi resource free karna ho (jaise timer stop karna ya event listener remove karna), to yeh kaam hota hai.

Handle Array with list

If you want to display an array as a list in JavaScript (especially in React), you can use the .map() function.

Why use map() function, not for loop?

- map() returns a new array, making it useful for immutability.
- map() is shorter and cleaner than a for loop.

In React, map() is commonly used to return JSX elements dynamically.

```
const fruits = ["Apple", "Banana", "Cherry"];
const fruitList = fruits.map((fruit) => <li key={fruit}>{fruit}</li>);

import { Table } from 'react-bootstrap'

function App() {
  const students = [
    {name:"Amit", email:"amit@test.com", mob:'111'},
    {name:"Sumit", email:"sumit@test.com", mob:'222'},
    {name:"Anil", email:"anil@test.com", mob:'333'},
  ]

  return (
    <>
    <h2>Array Handling</h2>
    <Table>
      <thead>
      <tr>
        <td>ID</td>
        <td>Name</td>
        <td>Email</td>
        <td>Mobile</td>
      </tr>
      </thead>
      <tbody>
      {
        students.map((data,index)=>
        <tr key={index}>
          <td >{index+1}</td>
          <td>{data.name}</td>
          <td>{data.email}</td>
          <td>{data.mob}</td>
        </tr>
        )
      }
      </tbody>
    </Table>
    </>
  )
}

export default App
```

Handle array nested list

```
import { Table } from 'react-bootstrap'

function App() {
  const students = [
    {
      name: "Amit", email: "amit@test.com", address: [
        { Hno: "37", city: "prayagraj", country: "india" }]
    },
    {
      name: "Sumit", email: "sumit@test.com", address: [
        { Hno: "42", city: "noida", country: "india" }]
    },
    {
      name: "Anil", email: "anil@test.com", address: [
        { Hno: "85", city: "gongaon", country: "india" }]
    },
  ]

  return (
    <>
      <h2>Array Handling</h2>
      <Table striped bordered variant='dark'>
        <thead>
          <tr>
          <td>ID</td>
          <td>Name</td>
          <td>Email</td>
          <td colSpan={3}>Address</td>
        </tr>
      </thead>
      <tbody>
        {
          students.map((data, index) =>
            <tr key={index}>
              <td>{index+1}</td>
              <td>{data.name}</td>
              <td>{data.email}</td>
              <td>{data.address[0].Hno}</td>
              <td>{data.address[0].city}</td>
              <td>{data.address[0].country}</td>
            </tr>
          )
        }
      </tbody>
    </Table>

  </>
 )
}

export default App
```

React Fragment (<></> or <React.Fragment>)

React Fragment ek lightweight wrapper hai jo multiple elements ko return karne me madad karta hai bina extra DOM node create kiye.

Why Use React Fragment?

a. Agar aap multiple elements return karte ho bina kisi parent wrapper (<div>) ke, toh error aayega
b. Using (<div>) not recommmended always - Ye extra <div> create karega jo unnecessary ho sakta hai.

Full Syntax (<React.Fragment>)

Agar aap keys ya attributes dena chahte ho toh full syntax use kar sakte ho

When to Use React Fragment?

- Jab multiple elements return karne ho bina extra <div> create kiye.
- Jab semantic HTML maintain karna ho.
- Jab performance optimize karni ho.

Lifting State Up (Child to Parent Data Passing)

Jab hume child component se parent component me data bhejna hota hai, toh callback function ka use kiya jata hai. Isko lifting state up kehte hain.

{Parent Component}

```
import User from './User'

function App() {
  function data(surname){
    alert("Amit "+surname)
  }

  return (
    <>
    <center>
      <User name={data}/>
    </center>
    </>
  )
}
export default App
```

{Child Component}

```
function User({name}){
    const surname = "Singh"
    return (
        <div>
            <h1>Hello </h1>
            <button onClick={()=>name(surname)}>Click</button>
        </div>
    )
}

export default User
```

Kaise Kaam Karta Hai?

- Parent component me ek function define hota hai jo data receive karega.
- Ye function child component ko props me diya jata hai.
- Child component me button click hone par ye function call hota hai aur data parent ko bhej diya jata hai.
- Parent component received data ko update karta hai.

Pure Component

Pure Component is a class component that only re-renders when there is a change in state or props. It is optimized to prevent unnecessary renders by implementing shallow comparison on props and state.

Key Points:

- Extends React.PureComponent instead of React.Component.
- Automatically implements shouldComponentUpdate() with a shallow comparison.
- Prevents re-renders if props and state remain the same.

Problem:

(In this code count not update but react always render this component which reduce the performance)

```
import React, { PureComponent } from "react";

class MyComponent extends React.Component {
  constructor(){
    super();
    this.state = {
      count:1
    }
  }
  render() {
    console.log("re-render")
    return (
      <>
      <h1>Amit, {this.state.count}</h1>
      <button onClick={()=>this.setState({count:1})}>Update Count</button>
      </>
    )
  }
}
export default MyComponent;
```

Solution:

(This code give the solution of that problem with Pure Component featured unnecessary re-rendering)

```
import React, { PureComponent } from "react";

class MyComponent extends PureComponent {
  constructor(){
    super();
    this.state = {
      count:1
    }
  }
  render() {
    console.log("re-render")
    return (
      <>
      <h1>Amit, {this.state.count}</h1>
      <button onClick={()=>this.setState({count:1})}>Update Count</button>
      </>
    )
  }
}

export default MyComponent;
```

useMemo

`useMemo` is a **React Hook** that **memoizes** the result of a computation to prevent unnecessary recalculations on every render. It **only recomputes the value when its dependencies change**, improving performance.

Problem:

(When item update there is no role of count but multicount function always execute which is unnecessary re-rendering)

```
import React, { useState } from ‘react’
 function App(){
      const [count, setCount] = useState(0)
```

```
     const [item, setItem] = useState(10)

     function multiCount(){
          console.log(“multicount”)
          return count * 5;
    }
    return (
          <div>
               <h1>useMemo hook in React </h1>
               <h2>Count : {count} </h2>
               <h2>Item : {item} </h2>
               <h2> {multiCount()} </h2>
               <button onClick = {()=> setCount(count+1)}>Update Count</button>
               <button onClick = {()=> setItem(item*10)}>Update Item</button>
          </div>

    )
}

export default App
```

Solution:
(We use useMemo hook It only recomputes the value when its dependencies change)

```
import React, { useState, useMemo } from ‘react’
 function App(){
     const [count, setCount] = useState(0)
     const [item, setItem] = useState(10)

    const multiCountMemo = useMemo(function multiCount(){
          console.log(“multicount”)
          return count * 5;
    },[count]);

    return (
          <div>
               <h1>useMemo hook in React </h1>
               <h2>Count : {count} </h2>
               <h2>Item : {item} </h2>
```

```
            <h2> {multiCountMemo }</h2>
            <button onClick = {()=> setCount(count+1)}>Update Count</button>
            <button onClick = {()=> setItem(item*10)}>Update Item</button>
        </div>

    )
}

export default App
```

useRef

`useRef` is a React Hook that allows you to:

- Access DOM elements directly (like input, button, div).
- Store values that don't trigger re-renders.
- Pass a reference to a child component (forwardRef).

You can use `useRef` to directly **access an input field** without updating state.

```
import React, { useRef } from "react";

const InputFocus = () => {
  const inputRef = useRef(null); // Creating a ref

  const focusInput = () => {
     console.log(inputRef.current.value)
     console.log(inputRef.current.style.color='red')
    inputRef.current.focus(); // Focuses the input field
  };

  return (
    <div>
      <input type="text" ref={inputRef} /> {/* Attach ref */}
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
};

export default InputFocus;
```

Unlike `useState`, `useRef` **can store values without causing re-renders**.

```
import React, { useRef, useState } from "react";

const Counter = () => {
  const [count, setCount] = useState(0);
  const refValue = useRef(0); // This does NOT trigger re-render

  const increment = () => {
    setCount(count + 1); // Triggers re-render
    refValue.current += 1; // Updates but does NOT re-render
    console.log("Ref Value:", refValue.current);
  };

  return (
    <div>
      <h1>State Count: {count}</h1>
      <h2>Ref Count (No Re-Render): {refValue.current}</h2>
      <button onClick={increment}>Increment</button>
    </div>
  );
};

export default Counter;
```

forwardRef

we use **`forwardRef`**, which allows **a parent component to pass a `ref` to a child component**.

`Child.js` (Child Component)

```
import React, { forwardRef } from "react";
const Child = forwardRef((props, ref) => {
  return <input type="text" ref={ref} placeholder="Enter text..." />;
});

export default Child;
```

`Parent.js` (Parent Component)

```
import React, { useRef } from "react";
import Child from "./Child"; // Importing Child Component

const Parent = () => {
  const inputRef = useRef(null); // Creating a ref

  const focusInput = () => {
    inputRef.current.focus(); // Focus the input field inside Child
  };

  return (
    <div>
      <Child ref={inputRef} /> {/* Passing ref to Child */}
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
};

export default Parent;
```

Controlled Component

Jis component me state ka use hota hai.

A **Controlled Component** in React means **the component's state is controlled by React** instead of the DOM.

- The **`value`** of an input field comes from `state`.
- The **`onChange`** event updates the state whenever the user types.

```
import React, { useState } from "react";

const ControlledInput = () => {
  const [text, setText] = useState(""); // State to store input value

  const handleChange = (event) => {
    setText(event.target.value); // Update state on user input
  };

  return (
    <div>
      <input type="text" value={text} onChange={handleChange} />
      <p>Typed Text: {text}</p>
    </div>
  );
};

export default ControlledInput;
```

⍰ The form fields **update state in real-time**.

⍰ **Handles multiple inputs** using a **single `onChange` function**.

⍰ Prevents default form submission with `event.preventDefault()`.

Uncontrolled Component

An **Uncontrolled Component** in React means **the component's state is controlled by the DOM**, not by React's state.

- The input field's value is **not stored in state**.
- We use **`useRef`** to access the input value **when needed** instead of tracking every change.

```
import React, { useRef } from "react";

const UncontrolledInput = () => {
  const inputRef = useRef(null); // Creating a ref for the input field

  const handleSubmit = () => {
    console.log("Entered Value:", inputRef.current.value); // Get input value directly
  };

  return (
    <div>
      <input type="text" ref={inputRef} /> {/* No state, uses ref */}
      <button onClick={handleSubmit}>Get Value</button>
    </div>
  );
};

export default UncontrolledInput;
```

- The **value is controlled by the DOM**, not React's state.
- Clicking **"Get Value"** logs the input's value **without re-rendering**.

Higher Order Function

```
import './App.css';
import React, { useRef, useState } from 'react'
function App() {
 return (
  <div className="App">
   <h1>HOC </h1>
   <HOCRed cmp={Counter} />
   <HOCGreen cmp={Counter} />
   <HOCBlue cmp={Counter} />
  </div>
 );
}
function HOCRed(props)
{
 return <h2 style={{backgroundColor:'red',width:100}}>Red<props.cmp /></h2>
}
function HOCGreen(props)
{
 return <h2 style={{backgroundColor:'green',width:100}}>Grren<props.cmp /></h2>
}
function HOCBlue(props)
{
 return <h2 style={{backgroundColor:'blue',width:100}}>blue <props.cmp /></h2>
}
function Counter()
{
 const [count,setCount]=useState(0)
 return<div>
  <h3>{count}</h3>
  <button onClick={()=>setCount(count+1)}>Update</button>
 </div>
}

export default App;
```

React Router

Introduction to React Router

What is React Router?

React Router is a **library** for handling navigation in React applications. It allows you to create **single-page applications (SPAs)** where different components are displayed based on the URL **without reloading the page**.

Why Use React Router?

- Enables navigation between different pages/components.
- Provides **client-side routing**, improving performance.
- Supports **dynamic routing** (URLs with parameters).
- Allows **nested routes** for better structuring.
- Helps create **protected routes** (for authentication-based access).

How React Router Works

React Router uses:

- **BrowserRouter**: Wraps the app and enables routing.
- **Routes**: Defines different paths in the application.
- **Route**: Maps a URL path to a React component.
- **Link & NavLink**: Used for navigation without refreshing the page.

Basic Routing in React Router

React Router provides three main components for routing:

- **BrowserRouter** – Wraps the app to enable routing.
- **Routes** – Groups all route definitions.
- **Route** – Defines a single route (path → component).

Setting up BrowserRouter

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App";

ReactDOM.createRoot(document.getElementById("root")).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Defining Routes with `<Routes>` and `<Route>`

```
import { Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/Contact";
function App() {
 return (
  <Routes>
   <Route path="/" element={<Home />} />
   <Route path="/about" element={<About />} />
   <Route path="/contact" element={<Contact />} />
  </Routes>
 );
}
export default App;
```

Linking in React Router (Link, NavLink, useNavigate)

To navigate between pages **without reloading**, React Router provides:

1. **`<Link>`** – Replaces `<a>` to prevent full-page reloads.
2. **`<NavLink>`** – Like `<Link>` but adds an "active" class when the link is selected.
3. **`useNavigate()`** – A hook to navigate programmatically.

1. Using `<Link>` for Navigation

Instead of `<a href="/">`, use `<Link to="/">`.

```
import { Link, Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/Contact";

function App() {
 return (
  <>
   <nav>
    <Link to="/">Home</Link> |
    <Link to="/about">About</Link> |
    <Link to="/contact">Contact</Link>
   </nav>

   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
    <Route path="/contact" element={<Contact />} />
   </Routes>
  </>
 );
}

export default App;
```

2. Using `<NavLink>` for Active Styling

Use `<NavLink>` instead of `<Link>` to style the active link automatically.

```
import { NavLink, Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/Contact";

function App() {
  return (
    <>
      <nav>
        <NavLink to="/" className={({ isActive }) => isActive ? "text-blue-500 font-bold" : ""}>
          Home
        </NavLink> |
        <NavLink to="/about" className={({ isActive }) => isActive ? "text-blue-500 font-bold" : ""}>
          About
        </NavLink> |
        <NavLink to="/contact" className={({ isActive }) => isActive ? "text-blue-500 font-bold" : ""}>
          Contact
        </NavLink>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </>
  );
}

export default App;
```

3. Navigating with `useNavigate()`

`useNavigate()` allows you to navigate **programmatically**.

```
import { useNavigate } from "react-router-dom";

export default function About() {
  const navigate = useNavigate();

  return (
    <div>
      <h1>About Us</h1>
      <button onClick={() => navigate("/")}>Go Back Home</button>
    </div>
  );
}
```

React Router Dynamic Routing (URL Params & useParams)

Dynamic Routing Kya Hota Hai?

Dynamic routing ka matlab hai ki ek **single component multiple pages ko handle kar sakta hai** by extracting values from the URL.

Example Use Cases:

- **User Profile Pages:** /user/101, /user/102, etc.
- **Product Pages:** `/product/iphone`, `/product/samsung`, etc.

Agar **har user ke liye alag page** (`User101.jsx`, `User102.jsx`) banane lage to **bohot duplicate code ho jayega**.
Iska **solution** hai **dynamic routing**, jisme ek **single component** (`UserProfile.jsx`) **har URL ke liye kaam karega**.

Kaise Define Karein Dynamic Route?

Sabse pehle `App.jsx` me ek **dynamic route** add karenge jo `:id` ke basis pe page load karega.

```
import { Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import About from "./pages/About";
import Contact from "./pages/Contact";
import UserProfile from "./pages/UserProfile";  // New Component

function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/contact" element={<Contact />} />
      <Route path="/user/:id" element={<UserProfile />} />  {/* Dynamic Route */}
    </Routes>
  );
}

export default App;
```

- **`:id`** ek placeholder hai, jo dynamic value lega.
- `/user/101`, `/user/102`, `/user/xyz` → Sab same `UserProfile.jsx` component use karenge.
- `id` ki value change hoti rahegi based on URL.

URL Parameters Ko Get Kaise Karein?

URL me jo `:id` pass ho raha hai, usko extract karne ke liye `useParams()` hook use karenge.

Create `pages/UserProfile.jsx`

```
import { useParams } from "react-router-dom";

export default function UserProfile() {
  const { id } = useParams();  // URL se "id" extract karenge

  return (
    <div>
      <h1>User Profile Page</h1>
      <p><strong>User ID:</strong> {id}</p>
    </div>
  );
}
```

- **`useParams()`** React Router ka hook hai jo URL se dynamic part extract karta hai.
- Agar URL `/user/101` hai, to `id = 101`.
- Agar URL `/user/xyz` hai, to `id = xyz`.

Navigation Kaise Karein Dynamic Routes Pe?

Ab `Home.jsx` page me **`<Link>`** ka use karenge taki different user profiles pe navigate kar sakein.

```
import { Link } from "react-router-dom";

export default function Home() {
  return (
    <div>
      <h1>Welcome to Home Page</h1>
      <h2>Users:</h2>
      <ul>
        <li><Link to="/user/101">User 101</Link></li>
        <li><Link to="/user/102">User 102</Link></li>
        <li><Link to="/user/abc">User ABC</Link></li>
      </ul>
    </div>
  );
}
```

- Ab User 101 pe click karoge to **`/user/101`** open hoga.
- User 102 pe click karoge to **`/user/102`** open hoga.
- User ABC pe click karoge to **`/user/abc`** open hoga.
- Sabhi cases me `UserProfile.jsx` hi load hoga, lekin `id` change ho jayegi.

React Router me `searchParams` kya hai?

React Router v6+ me, agar tumhe URL ke query string (`?key=value`) se data read ya set karna ho, to tum use karte ho:

import { useSearchParams } from "react-router-dom";

`useSearchParams()` Hook

const [searchParams, setSearchParams] = useSearchParams();

- `searchParams` → read karne ke liye
- `setSearchParams` → update/set karne ke liye

Step-by-Step Example

Step 1: URL with Search Params
http://localhost:5173/?name=raj&age=21

Step 2: React Component

```
import { useSearchParams } from "react-router-dom";

function UserInfo() {
  const [searchParams, setSearchParams] = useSearchParams();

  const name = searchParams.get("name");
  const age = searchParams.get("age");

  return (
    <div>
      <h1>Name: {name}</h1>
      <h2>Age: {age}</h2>

      <button onClick={() => setSearchParams({ name: "priya", age: "25" })}>
        Update Query Params
      </button>
    </div>
  );
}

export default UserInfo;
```

REST API in React

REST API Kya Hai?

REST API ek backend service hoti hai jo client (jaise React app) se request leti hai aur response bhejti hai. Yeh mainly HTTP methods use karti hai.
React frontend hota hai – dikhane ka kaam karta hai.
Lekin **data kahin aur se aata hai** — jaise backend API.

Hum API se kya karte hain?

Kaam	HTTP Method	Example
Data read karna	GET	/users
Naya data bhejna	POST	/users
Data update karna	PUT	/users/1
Data delete karna	DELETE	/users/1

API Call using `fetch` (Simple method)

Example: Users list fetch karna

```
import React, { useEffect, useState } from "react";

function App() {
  const [users, setUsers] = useState([]);
  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")  // GET request
      .then(res => res.json())
      .then(data => setUsers(data));
  }, []);
  return (
    <div>
      <h1>User List</h1>
      <ul>
        {users.map(u => (
          <li key={u.id}>{u.name}</li>
        ))}
      </ul>
    </div>
  );
}
export default App;
```

API Call using `axios` (Recommended)

```
import React, { useEffect, useState } from "react";
import axios from "axios";

function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    axios.get("https://jsonplaceholder.typicode.com/users")
      .then(res => setUsers(res.data));
  }, []);

  return (
    <div>
      <h2>Users</h2>
      {users.map(user => (
        <p key={user.id}>{user.name}</p>
      ))}
    </div>
  );
}

export default App;
```

POST Request – Naya data bhejna

```
const newUser = { name: "Ravi", email: "ravi@example.com" };

axios.post("https://jsonplaceholder.typicode.com/users", newUser)
  .then(res => console.log(res.data));
```

PUT Request – Update data

```
const updatedUser = { name: "Updated Ravi", email: "ravi@new.com" };

axios.put("https://jsonplaceholder.typicode.com/users/1", updatedUser)
  .then(res => console.log(res.data));
```

DELETE Request – Data delete karna

```
axios.delete("https://jsonplaceholder.typicode.com/users/1")
  .then(() => console.log("User deleted"));
```

Khud ki API se data fetch karna

1. GET Request – API se data fetch karna

```
import React, { useEffect, useState } from "react";

function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch("http://localhost:5000/users")  // <-- Replace with your API
      .then(res => res.json())
      .then(data => setUsers(data))
      .catch(err => console.log("Error:", err));
  }, []);

  return (
    <div>
      <h1>User List</h1>
      <ul>
        {users.map(u => (
          <li key={u.id}>{u.name}</li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

2. POST Request – Naya data bhejna

```
function addUser() {
  const newUser = {
    name: "Ravi",
    email: "ravi@example.com"
  };

  fetch("http://localhost:5000/users", {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(newUser)
  })
    .then(res => res.json())
    .then(data => console.log("User Added:", data))
    .catch(err => console.log("Error:", err));
}
```

3. PUT Request – Data update karna

```
function updateUser() {
  const updatedUser = {
    name: "Updated Ravi",
    email: "ravi@new.com"
  };

  fetch("http://localhost:5000/users/1", {
    method: "PUT",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify(updatedUser)
  })
    .then(res => res.json())
    .then(data => console.log("User Updated:", data))
    .catch(err => console.log("Error:", err));
}
```

DELETE Request – Data delete karna

```
function deleteUser() {
  fetch("http://localhost:5000/users/1", {
    method: "DELETE"
  })
    .then(() => console.log("User Deleted"))
    .catch(err => console.log("Error:", err));
}
```